

# Dual en



# avanade

---

# Avanade

---

Es una empresa internacional que se dedica ofrecer distintos tipos de servicios, desde asesoría, creación de aplicaciones empresariales como soluciones CRM y ERP, soluciones en Cloud, digitalización y servicios tecnológicos.

Sus soluciones empresariales se basan en la plataforma Microsoft Dynamics 365.

Avanade surge en el 2000 fundada entre Accenture y Microsoft. Su principal sede se encuentra en Seattle, Washington, EE.UU.

Destacar en 2017 el premio a España de Microsoft Partner.



# Formación inicial en la empresa

---

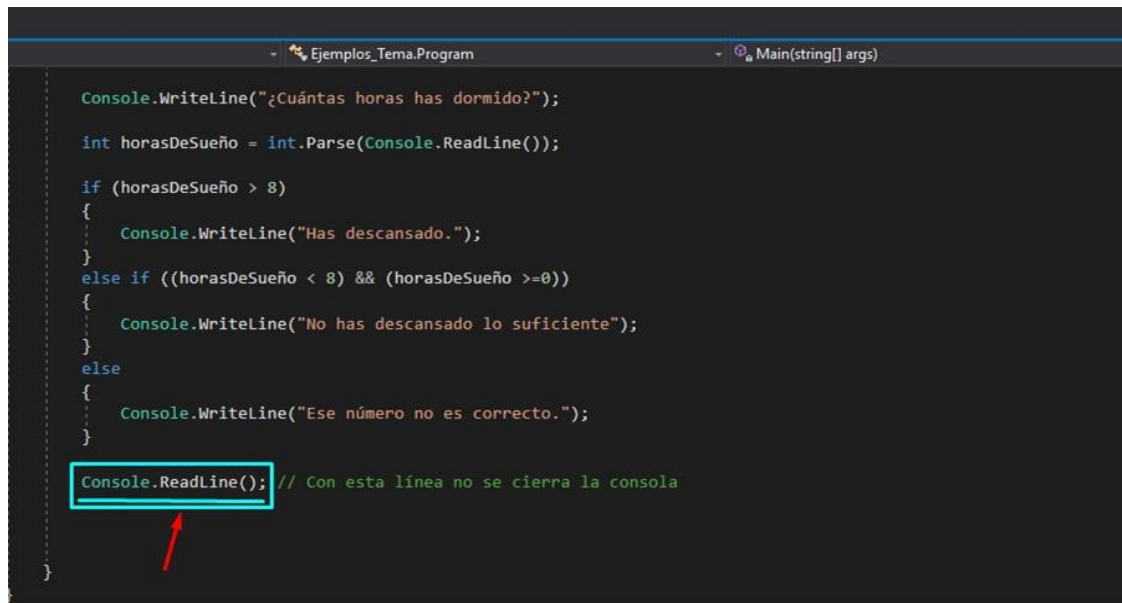
Hemos realizado un curso de C# Station a través de la página web [C#Station](#). También, en la misma web realizamos el curso de ADO.Net mediante el cual aprendimos a incluir bases de datos realizadas en SQL Server a nuestro código programado en C# en Visual Studio.

Además de hacer los cursos, hemos elaborado distintas prácticas llevadas a cabo en consola. Así, programamos pequeños ejercicios de bucles, arrays, listas, etc para familiarizarnos con el lenguaje y ver las diferencias que hay en c# respecto a java.

# Aplicación de consola

A la hora de realizar una aplicación en consola en c# debemos incluir una línea para que no se cierre esta al terminar de ejecutarse.

- `Console.ReadLine();`



```

Ejemplos_Tema.Program
Main(string[] args)

Console.WriteLine("¿Cuántas horas has dormido?");

int horasDeSueño = int.Parse(Console.ReadLine());

if (horasDeSueño > 8)
{
    Console.WriteLine("Has descansado.");
}
else if ((horasDeSueño < 8) && (horasDeSueño >=0))
{
    Console.WriteLine("No has descansado lo suficiente");
}
else
{
    Console.WriteLine("Ese número no es correcto.");
}

Console.ReadLine(); // Con esta línea no se cierra la consola
}

```

# Aplicaciones en Winform

---

A la hora de realizar una aplicación en winform estructuramos el código en 3 capas:

- ❖ Acceso a base de datos
- ❖ Capa lógica
- ❖ La interfaz en este caso son los formularios

Llamamos de una capa a otra a partir de métodos.

# Prácticas

---

De forma adicional a la aplicación que hemos realizado en Winform y que ha sido, como decía Alejandro, el núcleo de nuestra formación, durante el transcurso de la misma también realizamos algunas prácticas con un enfoque más acotado, pero revisando no obstante conceptos fundamentales del desarrollo en C#.

De estas prácticas, realizamos un total de 5:

- Persons
- Concessionaire
- School
- VideoClubManagement
- AvaStore

# 1. Conceptos teóricos. ¿En qué consistían estas prácticas?

---

Normalmente, compartían uno o varios factores comunes. El objetivo de estas prácticas era reforzar los conceptos que hemos venido aprendiendo en el instituto durante los primeros meses con algo más de especificación en ciertos aspectos. Es por ello por lo que estos ejercicios apuntaban todos a una misma dirección.

A continuación se extraen los conceptos clave de estos ejercicios.

# 1.1. División por capas

---

De los puntos más interesantes y novedosos que aprendemos podemos destacar **la estructuración de la aplicación en distintas capas**, técnica también conocida como **arquitectura multinivel**. En el caso de la aplicación principal, OrdersManagement, lo que hacíamos era dividir el núcleo del programa en tres capas: interfaz de usuario, business logic y data access. ¿Qué se consigue con esto? Lograr una **mejor organización del código**, una **mayor eficiencia** (un ejemplo podría ser el traspaso de excepciones para ser tratadas por una única capa receptora mediante un *throw*), una **simplificación de código y de la comprensión** y una **mantenibilidad** mucho **más sencilla**.

Esta última característica es, de hecho, la más relevante dentro de las mencionadas, ya que implica una abstracción de cada uno de los niveles, facilitando el desarrollo a posteriori o las modificaciones que se pretendan insertar.



## 1.2. Visibilidad

---

En C#, tenemos cinco opciones distintas.

En el 90% de los casos se usarán los modificadores *private* o *public*. En algunos casos específicos se usarán *protected*, *protected internal* o *internal*.

Access Modifier	Description (who can access)
private	Only members within the same type. (default for type members)
protected	Only derived types or members of the same type.
internal	Only code within the same assembly. Can also be code external to object as long as it is in the same assembly. (default for types)
protected internal	Either code from derived type or code in the same assembly. Combination of protected OR internal.
public	Any code. No inheritance, external type, or external assembly restrictions.

## 1.3. Comentarios

---

Una utilidad muy interesante del Visual Studio Code es la sencilla inserción de comentarios sobre propiedades, métodos o clases. Lo único que tenemos que hacer es introducir tres veces el símbolo “/”, de modo que se autogenera una plantilla lista para rellenar. Encontramos diversos campos, como *summary*, *params*, o *return*. En el primero introducimos un breve resumen, en el segundo los parámetros, y en el tercero, si devuelve algo, explicamos cómo lo hace.

```
/// <summary>
/// Add a person to the list.
/// </summary>
/// <param name="Person">Person.</param>
/// <returns>True if person was added.</returns>
public bool AddPerson(Person Person)
{
    bool added = false;
    bool available = false;
    int availablePosition = 0;
```

## 1.4. Propiedades

---

Podríamos decir que las propiedades serían un punto intermedio entre los atributos/campos y los métodos de una clase. ¿Por qué? Porque comparten las funciones de uno y otro. Son miembros de la clase a los cuales se les asigna un valor, pero que, a su vez, permiten la modificación y establecimiento de dichos valores mediante *get* y *set*. Esto, lógicamente, nos facilita la vida muchísimo. Su declaración sería del siguiente modo:

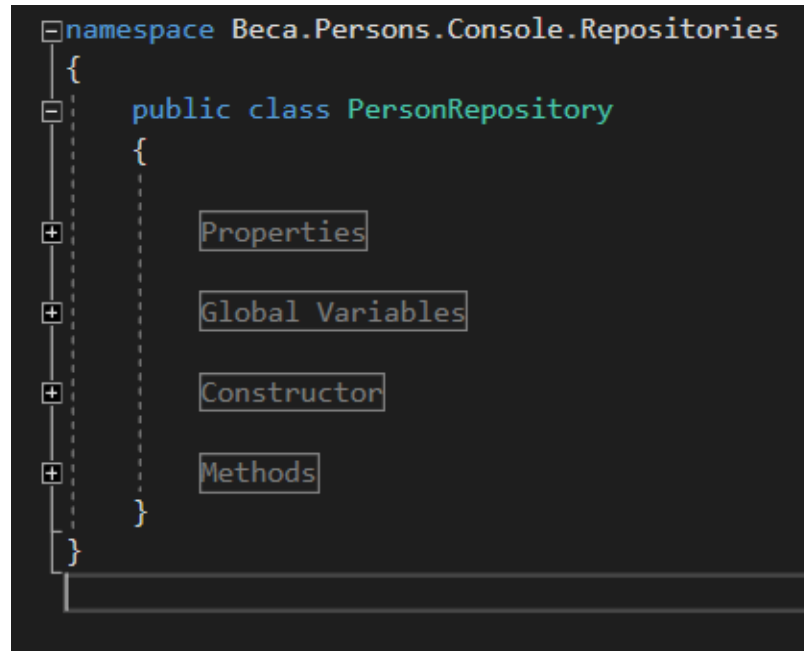
```
public string name { get; set; }
```

# 1.5. Regiones

---

Podríamos decir que son organizadores de código. La sintaxis funciona de la siguiente manera: colocamos “#region” + nombre al principio y “#endregion” + nombre al final. De esta manera, el entorno nos permite ampliar o reducir el código a nuestro parecer.

Lo vemos reducido:



```
namespace Beca.Persons.Console.Repositories
{
    public class PersonRepository
    {
        Properties
        Global Variables
        Constructor
        Methods
    }
}
```

The image shows a code editor with a dark background. On the left side, there is a vertical toolbar with icons for folding and unfolding code regions. The code itself is written in C# and defines a namespace `Beca.Persons.Console.Repositories` containing a `PersonRepository` class. The class body contains four sections: `Properties`, `Global Variables`, `Constructor`, and `Methods`. Each of these sections is enclosed in a dashed-line box, indicating they are collapsed regions. The `Properties` and `Global Variables` sections are currently collapsed, while the `Constructor` and `Methods` sections are expanded.

# 1.5. Regiones

Y ampliado:

```
namespace Beca.Persons.Console.Repositories
{
    public class PersonRepository
    {
        #region Properties

        /// <summary>
        /// Number of objects the array contains.
        /// </summary>
        public int count { get { return _count; } }

        /// <summary>
        /// Array size.
        /// </summary>
        private int length { get; set; }

        #endregion Properties

        Global Variables

        Constructor

        Methods
    }
}
```

## 1.6. Herencia

---

Hemos utilizado muy a menudo clases derivadas de otras superiores que permitían la creación de clases hijas de una manera más cómoda y efectiva. De este modo, colocábamos o establecíamos una serie de propiedades comunes en la clase padre y en las derivadas especificábamos las correspondientes a cada clase. Es el caso de la práctica *Persons*, donde aplicábamos la herencia en tres de las clases creadas.

# 1.7. Sobrecarga

---

Repasábamos también de forma constante las variaciones de distintos métodos que parten de uno raíz. Muy a menudo, aplicábamos esta técnica en la definición de las clases.

## 1.8. Colofón

---

Mencionar, ya por último para no extendernos demasiado, algunos “trucos” o “atajos” que nos han sido de gran utilidad a la hora del desarrollo de las prácticas y que resultan muy curiosos:

- **Booleans condicionales en una sola línea:** se instancia un campo de tipo *bool* que se establece como true si la condición colocada a la derecha también lo es.

Ejemplo:

```
bool userValidated = PersonRepository.CheckUser() > 0
```

- **Sustitución de arrays por listas dinámicas:** en todo momento trabajamos con listas dinámicas y no con arrays, precisamente por las limitaciones y problemas que generan estos últimos al tener un índice inamovible.



## 1.8. Colofón

---

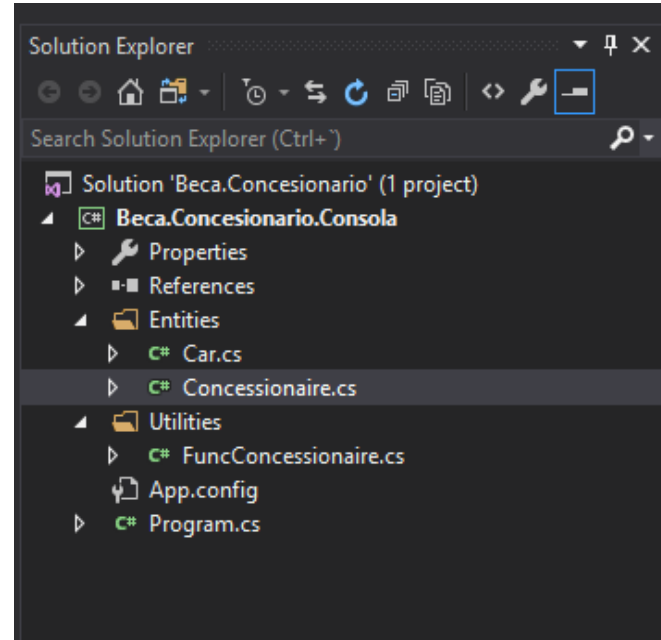
- **Manejo y aplicación de bloques try/catch:** estudiamos no de una forma demasiado exhaustiva el uso de bloques try/catch para el manejo de excepciones, incluyendo también el *finally* en algunos casos.
- **Creación de propiedades readonly:** aprendemos también a cómo crear este tipo de propiedades para aquellos casos en los que carezca de sentido que el usuario pueda cambiar el valor.

Estos serían quizás los más recurrentes, si bien es cierto que hemos estudiado y tratado otros conceptos. No obstante, debido a su mayor complejidad y al carácter específico de algunos de ellos, no los hemos usado en la realización de nuestras prácticas. Hablamos en este sentido de los **delegados**, **métodos anónimos** o el uso de **structs** en lugar de clases, por citar algunos ejemplos.

# 1.9. Ejemplo práctico - Concessionaire

---

Práctica consistente en la creación un programa para la gestión de coches en un concesionario.



Observamos la organización en distintas capas: por un lado tenemos las clases, que permiten instanciar los correspondientes objetos; y por otro lado otra clase que nos permite interactuar entre dichos objetos.

A la derecha podemos observar la definición de *Car*.

```
namespace Beca.Concesionario.Consola.Entities
{
    /// <summary>
    /// Car class definition.
    /// </summary>
    public class Car
    {
        /// <summary>
        /// Brand.
        /// </summary>
        public string Brand { get; set; }

        /// <summary>
        /// Model.
        /// </summary>
        public string Model { get; set; }

        /// <summary>
        /// Engine measure.
        /// </summary>
        public int Horsepower { get; set; }

        /// <summary>
        /// External design.
        /// </summary>
        public string Coating { get; set; }
    }
}
```

```
public Car()
{
    this.Brand = string.Empty;
    this.Model = string.Empty;
    this.Horsepower = 0;
    this.Coating = string.Empty;
}

/// <summary>
/// Constructor.
/// </summary>
/// <param name="brand">Brand.</param>
/// <param name="model">Model.</param>
public Car(string brand, string model)
{
    this.Brand = brand;
    this.Model = model;
}

/// <summary>
/// ToString object method override.
/// </summary>
/// <returns>String with main properties</returns>
public override string ToString()
{
    return Brand + " " + Model + " " + Horsepower + " " + Coating;
}
}
```

En la primera parte veíamos la definición de las propiedades y en esta otra imagen observamos la de los constructores.

A continuación, mostramos la definición de *Concessionaire*.

```

namespace Beca.Concesionario.Consola.Entities
{
    /// <summary>
    /// Concesionario class definition.
    /// </summary>
    public class Concessionaire
    {
        /// <summary>
        /// Name.
        /// </summary>
        public string Name { get; }

        /// <summary>
        /// Brand.
        /// </summary>
        public string Brand { get; }

        /// <summary>
        /// City.
        /// </summary>
        public string City { get; }

        /// <summary>
        /// Cars list.
        /// </summary>
        public List<Car> Carslist { get; set; }
    }
}

```

```

    /// <summary>
    /// Cars quantity.
    /// </summary>
    public int CurrentCarsNumber { get; set; }

    /// <summary>
    /// Car models quantity.
    /// </summary>
    public int CurrentModelsNumber { get; set; }

    /// <summary>
    /// Constructor.
    /// </summary>
    public Concessionaire()
    {
        // Is this good in case we handle nulls?
        this.Name = "Volkswagen Benalmádena";
        this.Brand = "Volkswagen";
        this.City = "Málaga";
        this.CarsList = new List<Car>();
    }
}

```

Por último, la definición de  
*FuncConcessionaire*

```
/// <summary>
/// Obtains a cars list from a concessionaire object.
/// </summary>
/// <param name="concessionaire">Concessionaire object.</param>
/// <returns>A list of cars from that concessionaire.</returns>
public List<Car> ObtainCars(Concessionaire concessionaire)
{
    if (concessionaire.CarsList != null)
    {
        this.carsList = concessionaire.CarsList;
        return carsList;
    }
    else
    {
        this.carsList = null;
        return this.carsList;
    }
}
```

```
/// <summary>|
/// Finds which concessionaires are operative (have cars).
/// </summary>
/// <param name="concessionaireOne">First concessionaire object.</param>
/// <param name="concessionaireTwo">Second concessionaire object.</param>
/// <returns>List with all the concessionaires that have cars.</returns>
public List<Concessionaire> CountOperativeConcessionaires(Concessionaire concessionaireOne, Concessionaire concessionaireTwo)
{
    List<Concessionaire> concessionairesList = new List<Concessionaire>();
    concessionairesList.Add(concessionaireOne);
    concessionairesList.Add(concessionaireTwo);
    List<Concessionaire> operativeConcessionaires = new List<Concessionaire>();

    foreach (Concessionaire concessionaire in concessionairesList)
    {
        if (concessionaire.CarsList != null && concessionaire.CarsList.Count() > 0)
        {
            operativeConcessionaires.Add(concessionaire);
        }
    }
    return operativeConcessionaires;
}
```

## 2. Cursos en Plural Sight

---

Nuestro principal portal de aprendizaje. Se trata de eso, una plataforma con un contenido amplísimo y variado donde puedes realizar cursos para desarrollar tus distintas habilidades como programador. La **intención principal** de este sitio web es proporcionar mediante cursos, evaluaciones y técnicas de aprendizaje, ayuda tanto a colectivos como a individuos, sobre todo en lo referente a cubrir ciertas áreas críticas, a innovar de forma más acelerada y en cómo atacar los objetivos clave.

Las estadísticas hablan por sí solas acerca del éxito de la página:

- 1300 expertos
- Más de 6000 cursos
- Más de 800 empleados

Algunas de las tecnologías con las que trabaja son **Adobe, Oracle, Microsoft, Google, Unity o StackOverflow.**



## 2.1. ¿Qué podemos hacer en Plural Sight?

---

Cursos, cursos, cursos... La información que tenemos en el portal es muy buena, puesto que cubre un gran rango de tecnologías, con muchísima documentación para cada una de ellas.

En nuestro caso, profundizamos en dos cursos:

- C# Fundamentals with Visual Studio 2015
- C# Best Practices: Collection and Generics

A lo largo de los cursos no solo tenemos oportunidad de visualizar el contenido, sino que también podemos añadir marcas o notas, leer la transcripción completa de todo lo que dice el autor, compartir en redes sociales, descargar ejercicios relacionados con las explicaciones, foros para resolución de dudas... y un largo etc que completa todas las necesidades que el usuario pueda tener.

Hay tan solo un pequeñísimo detalle a tener en cuenta: **no es gratis. Cuesta solamente (nótese la ironía) 300 dólares** al año, con lo que si queremos usarlo tenemos que pasar por caja.

# Innova Learn y Webcast

---

Los *Innova Learn* son sesiones tecnológicas que se realizan en advance con el propósito de compartir conocimientos, y mantener al día a los empleados con las últimas novedades tecnológicas. En el tiempo de formación que hemos estado en la empresa hemos presenciado dos Innova Learn:

# Innova Learn SharePoint

---

Hemos presenciado varios innova learn muy interesantes, uno de ellos es sobre SharePoint Framework:

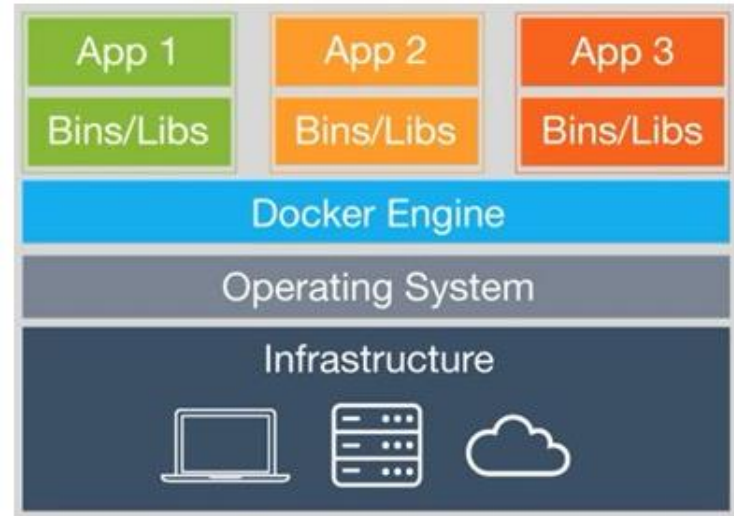
- Introducción a SharePoint Frameworks
- Últimas novedades introducidas
  - Demo de la nueva extensión
- Demo sobre como implementar un elemento web del lado cliente de SharePoint en una red CDN de Azure

# Innova Learn docker

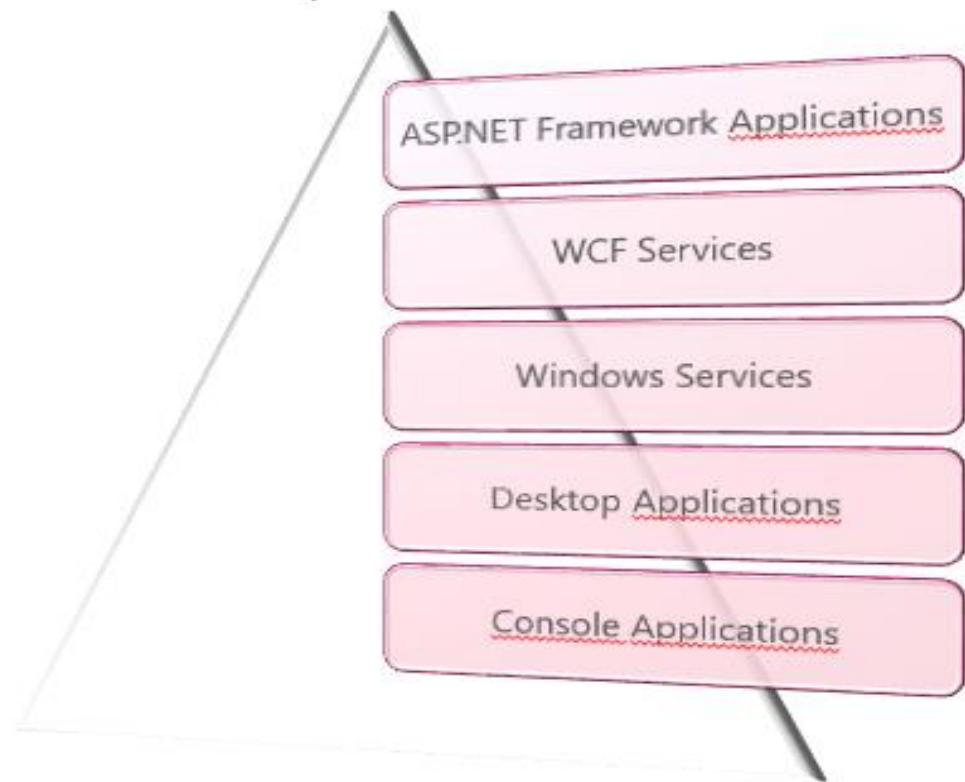
---

Otro innova learn de los que hemos visto es sobre docker, una forma de modernizar aplicaciones utilizando contenedores.

- Introducción a Docker
- Demo sobre como utilizarlo
- Instalación de docker para Azure
- Seguridad
- Monotorización
- Backups y restauración



# Tipos de Aplicaciones para ser containerizadas



# Herramientas y tecnologías utilizadas

---



# Valoración personal

---

Alejandro De La Maza

Lucía Flores

Iván Miranda

Beatriz Parejo